

Characterization of an Extended Multimedia Benchmark on a General Purpose Microprocessor Architecture

Seongwoo Kim and Arun K. Somani
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa 50011, USA

Abstract

Increasing demand for multimedia applications (MMAs) by general-purpose processor (GPP) users leads architects to merge multimedia processing capability into GPPs. To develop a GPP providing high quality multimedia services as well as high performance general computing, thorough understanding of the characteristics of both types of applications is fundamental. This paper presents an comprehensive multimedia benchmark suite with variable input data files, and quantifies the characteristics of the presented benchmark in comparison to SPEC integer and floating-point benchmarks. Various aspects of program behavior are measured using a run-time tracer. We identify several distinctive characteristics of MMAs compiled for a GPP and discuss the corresponding architectural impacts on GPPs.

1 Introduction

Multimedia applications (MMAs) are becoming dominant users of computer systems, and their marketplace is growing dramatically. Depending on the budget and application area, different classes of microprocessors are used. Dedicated processors such as digital signal processing (DSP) and embedded processors, which usually have very limited processing capabilities and programmability, are popular for cost-sensitive products [1], [2], [3]. Media processors implement the functionalities of several DSP processors on a single chip for high performance multimedia on low-cost PCs [4], [5]. In addition to these specialized processors, most state-of-the-art general-purpose processors (GPPs) used in personal computers and workstations have also included special hardware units and/or multimedia instruction extensions, enhancing performance of multimedia tasks to fulfill growing GPP users' need [6].

Despite increasing demand for multimedia processing on GPP-based systems, very little attention has been paid to the overall needs of MMAs. The research in GPPs has been mainly carried out for general integer and scientific floating-point programs, but there is a dearth of published work on quantitative analysis of multimedia workload characteristics. Even new architectures recently proposed for future GPPs in the literature have not been fully examined for MMAs. One reason for this is that there have been no widely accepted benchmark suites, such as SPEC CPU benchmarks [7], throughout the industry and the press.

Characterizing the workload is a fundamental step in microprocessor design. Based on the analysis of targeted applications, architects allocate chip area and determine processor's features. Different attributes of MMAs may require significant changes of GPP design. Thus, a sufficient analysis of multimedia workload needs to be done to effectively optimize multimedia computing using GPPs. In [2], Eyre and Bier showed that many GPPs enhanced with DSP extensions are not very efficient for MMAs in cost/performance aspects. The increased multimedia processing capability of these GPPs mostly results from high clock rate and hand-optimized assembly code. Current patchwork to GPP architecture to merge multimedia computing with general computing does not fully take advantage of inherent parallelism in MMAs. In order to achieve the performance improvement on overall applications, GPP architecture should be founded on thorough understanding of MMAs as well as general-purpose and scientific applications (GSAs).

The characteristics of MMAs are often derived from intuitive observations of the algorithms and structures of the programs. However, actual program behavior on a system can be considerably different. In this paper, we investigate run-time behavior of a large multimedia benchmark set that is collected from frequently used audio, image and graphics, and video programs in the public domain. The execution traces of 22 programs on a high-end GPP are examined and the observed characteristics are compared with well-known SPEC benchmark suites. Our study targets to make the following contributions.

- *A representative benchmark suite of MMAs is presented.* The diverse constituent programs with sufficiently long and variable inputs can serve as a comprehensive multimedia workload in the measurement and modeling of computing systems.
- *The characteristics of a wide range of MMAs are quantified.* This verifies rough knowledge about MMAs and facilitates the development of efficient architectural strategies. In the literature, such quantitative measurement the system architect can refer to is very limited.
- *An architectural guideline for multimedia processing enhancement of GPPs is proposed.* The suggestions are made based on our quantitative understanding of both MMAs and GSAs.

This paper is organized as follows. Section 2 discusses related work and places our study in the corresponding research space. Section 3 provides a detailed description of our benchmark set and experimental setup. The premise of the experiments is also given in this section. Section 4 presents an analysis of MMA program behavior in comparison to the behavior of GSAs. We use various metrics to identify influential characteristics on the GPP design. In Section 5, we present concluding remarks.

2 Related Work

An in-depth characterization of GSAs was given in [8]. This paper presents a similar study for MMAs. Several studies have examined the performance analysis of multimedia instruction extensions of GPPs for a specific class of MMAs [9], [10], [11]. Bhargava et al. [12] measured the effectiveness of Intel’s MMX [14] by comparing the performance of non-MMX version and MMX version of DSP kernels and multimedia applications. Instruction level profiling was used to identify the reasons for the observed speedup, ranging 1.2 to 7.5. Ranganshan et al. [13] compared the performance of image and video processing applications on simulated GPP models, without and with Sun’s VIS media ISA extensions. They showed that the combination of the VIS extension and multiple-issue out-of-order features provide an average factor of 5.5X performance improvement over the single-issue in-order processor model. However, they indicated that the VIS extension also limits the benefits on the benchmarks due to its overhead and fixed parallelism exploitation. While previous studies measured the limited benefit of multimedia instruction extensions, our study focus on quantifying intrinsic characteristics of MMAs comparing with GSAs on a GPP to provide reference data in exploring new architectural techniques for MMA support using GPPs.

Few attempts to quantify the distinctive nature of overall MMAs on GPPs exist in the literature. Balakrishnan and Garg collected a small multimedia benchmark suite and compared its performance on four GPP-based systems [15]. They found that a system’s performance in multimedia processing is hard to predict based on its integer performance, stressing the need for a multimedia benchmark.

Lee et al. presented a reasonable multimedia benchmark set called MediaBench [16]. The MediaBench is the first MMA collection in academia and its uniqueness was shown by comparing performance characteristics with SPEC integer programs. However, it includes only one video application although video programs constitute a major group of MMAs. Besides, its input files are short and monotonous, and some of media files generated (encoding after decoding) by benchmark programs are not playable. This motivates us to make a more comprehensive and reliable collection of benchmarks.

In 1998, SPEC’s graphics group formed the Multimedia Benchmark Committee (MBC) or SPECmedia [18] to develop a wide range of standardized multimedia benchmarks, but the group currently focuses only on MPEG-2 [17] and the benchmark has not been available yet. In [19], several ideas and methodologies for each aspect of multimedia performance benchmarking are presented.

Liao and Wolfe showed an extremely high degree of instruction level parallelism (ILP) available in video applications using an ambitious machine model [20]. Because of their unrealistic assumptions such as perfect branch prediction and code optimization, the uncertainty of program behavior on real machines could not be removed.

3 Benchmarks and Experimental Setup

Benchmark set: New MMAs are emerging and require a wide class of DSP functions that have been implemented in diverse ways. The fast changing field of MMA development has made it difficult to develop a wide range of standardized benchmark suites. It is, however, reasonable to assume that a substantially representative set of multimedia benchmarks can be collected at this time. Figure 1 presents a short description of our multimedia benchmark set¹. Several programs used in MediaBench [16] also appear in our benchmark, but all programs were collected from independent sources and the latest versions were used, if available. All outputs from the test runs using our input data set were examined to verify the correct functionalities of the benchmark.

¹The collected benchmark suite is available on our web site.

	Benchmark name used	Program source	Input files	Benchmark description
A u d i o	encode decode	G.721	speech*	An ANSI-C language reference implementation of the CCITT (International Telegraph and Telephone Consultative Committee) G.721 voice compression. ftp://svr-ftp.eng.cam.ac.uk/comp.speech/
	mpa_enc mpa_dec	MPEG audio 1.10	speech* music	MPEG-1 and MPEG-2 audio codec written by the MPEG audio software simulation group. www.mpeg.org
	rasta	RASTA 2.4	speech*	An implementation of some of the RASTA speech* recognition front end algorithms in addition to basic PLP processing. www.icsi.berkeley.edu/real/rasta.html
	rawaudio rawdaudio	ADPCM 1.2	speech*	A simple 16-bit PCM to/from 4-bit ADPCM coder and decoder. www.cwi.nl/~jack/
	toast untoast	GSM 6.10	speech*	Implementation of the European GSM 06.10 provisional standard for full-rate speech transcoding. GSM is used for cellular phone communications. kbs.cs.tu-berlin.de/~jutta/toast.html
I m a g e	epwic unepwic	EPWIC 1.0	complex image simple image	A wavelet based grayscale image compression utility written in C. www.cis.upenn.edu/~butch/EPWIC/index.htm
	jpeg	JPEG	vigo.ppm	A graphics compression and decompression program (from SPEC95). www.spec.org
	ppmtogif pgmoil	PBMPLUS	ppm image pgm image	A toolkit for converting various image formats to/from portable formats (ppmtogif: convert portable pixmap to GIF) and manipulating the portable formats (pgmoil: turn a portable graymap into an oil painting). www.acme.com/software/pbplus/
	pointblast reflect	MESA 3.0	not required	A 3-D graphics library with an API similar to OpenGL. www.mesa3d.org
V i d e o	mpeg_enc	MPEG-1 Encoder 1.5	field of flowers table tennis	A portable software-only MPEG-1 video decoder. MPEG is a compression standard for audio and video data established by the ITU and ISO. bmc.berkeley.edu/projects/mpeg/
	mpeg_play	MPEG-1 Player 3.2	musician playing dynamic scene	A portable software-only MPEG-1 video encoder. bmc.berkeley.edu/projects/mpeg/
	mpeg2play	MPEG-2 1.1b	slowly moving windmill rapid motion of cheerleaders	An MPEG2 video player based on the MPEG Software Simulation Group's reference code, but optimized for speed. MPEG-2 is the standard for digital television. www.mpeg.org/MPEG/MSSG/
	tmn tmndec	H.263 2.0	rapid motion person talking	A very low bitrate video codec for videoconferencing over analogue phone lines. This program is an ITU recommendation H.263 compatible video encoder and decoder written in C. www.fou.telenor.no/brukere/DVCI
	pgp	PGP 2.6	pgp encrypted text	A high-security cryptographic software application that allows people to exchange messages with both privacy and authentication. web.mit.edu/network/pgp.html

Figure 1: Description of our multimedia benchmark set. *The *speech* consists of two radio broadcasting files which include voice, music and audience's noise.

We also included other popular applications such as MPEG-1 video and MPEG-1/2 audio. In addition, two image processing programs, *ppmtogif* and *pgmoil*, were selected from PBMPLUS, and the encoder and decoder of a video standard H.263 were also added. A graphics compression/decompression program, *jpeg*, was taken from the SPEC95 integer suite. We classify the multimedia benchmark set into three parts, *audio*, *image* and *graphics*, and *video* groups as shown in Figure 1. In consideration of MMAs which require secured communication, we included an encryption program, called *pgp*.

Inputs to benchmarks: To test the impact of input files on program behavior, two input files of different natures are used. For example, one video file of *mpeg_play* shows very little change between adjacent frames such as a person talking while sitting on a chair. The other input file includes dynamic scene changes with different color modes. MPEG audio plays speech and music audio files. The processing of one image with a lot of color variation is compared with another image that includes a large white space. All input files are sufficiently large to ensure full demonstration of run-time program behavior. For comparison purposes, we also use all SPEC95 integer (SPECint) and floating-point (SPECfp) programs as representative benchmarks of GSAs, respectively. Input files used for SPEC benchmarks are given in Figure 2. To make each program run demonstrates full behavior, program inputs are sufficiently long and changeable, and two input files of different nature are associated with one program, if necessary.

Trace characterizers: To collect loop timing data, *loopreport/looptool* is used. The cycle-by-cycle traces of the programs are applied to the characterizers using Sun's Shade tool V5.33C [21]. The execution profiling is performed on-the-fly to eliminate large storage requirements for the trace files. This also enables us to examine sufficiently long executions of the programs. While up to 4 billion instructions are executed², desired information is dynamically

²Less than 4 billion instructions are executed for ADPCM, *gcc*, *untoast*, *tmndec*, *decode*, *pgp*, MESA, *rasta*.

SPECint		SPECfp	
Benchmark	Input file	benchmark	Input file
compress	bigtest.in	applu	applu.in
gcc	cp-decl.i	apsi	apsi.in
go	9stone21.in	fpppp	natoms.in
li	*.lsp	hydro2d	hydro2d.in
m88ksim	ctl.in	mgrid	mgrid.in
perl	primes.in	su2cor	su2cor.in
vortex	vortex.in	swim	swim.in
		tomcatv	tomcatv.in
		turb3d	turb3d.in
		wave5	wave5.in

Figure 2: SPEC95 suite and input files.

captured and redirected to the characterizers. It should be noted that an adequately large sample size of program execution is critical to obtain a better understanding of the characteristics of the entire program. Initial routines of the programs of a benchmark set may be similar, and they may take up millions of instructions. Many trace-driven simulations used in the literature consider less than a billion instruction executions.

Environment: Our evaluation is performed on Sun Ultra-5 workstations under the Solaris 2.5.1 operating system environment. All benchmarks' executable files are built using Sun Workshop Compilers, `cc` and `f77`, with compilation flags of `xO4`, `dalign`, and `xarch=v8`.

Although the platforms have an instruction extension for multimedia processing (i.e., VIS instruction set), it should be noted that we do not take specially optimized codes of the benchmarks using the multimedia instructions. The performance gain achieved by the VIS and its limitation have been already shown in the literature. There is no compiler that automatically generates a fully enhanced code with multimedia instructions from an MMA written in a high-level language. For a highly tailored code, hand-optimization is needed. In our study, the optimization target is not MMA codes. Thus, both GSAs and a new set of benchmarks, MMAs, are conventionally compiled with a high optimization level, and their exhibiting run-time behavior is quantified and analyzed to exploit better for GPP architecture optimization.

4 Analysis of Benchmark Characteristics

This section reports the nature of our multimedia benchmark set and provides an analysis with respect to a GPP design approach. characteristics in terms of the same metrics for three application groups, denoted by *SPECint*, *SPECfp*, and *MMA*, are compared. The average value over each group is given in the graph with the group name. The results for MMA are grouped by the type of input medium. Although two input files were examined for most MMA programs, both results are neither presented in graphs nor used for statistics unless they differ by more than 3.0 in percentage or 1.5 in instruction counts.

Figure 3 shows the arithmetic means of processing speeds in KIPS (thousands of instructions per second) for three benchmark groups. The KIPS can be a misleading metric, but we intend to obtain a rough idea of relative performance among the groups. Despite the expectation of high inherent parallelism in MMA, on average the processing speed for MMA is just a little bit better than the speed for SPECint. In the following sections, we find out the reasons for this by identifying the degree of difference in characteristics and the corresponding impact on the architecture. The more common characteristics found over different benchmarks, the easier it is for the architects to make optimal design choices for a balanced system.

Benchmark	SPECint	SPECfp	MMA
KIPS	7198.36	20861.85	8407.47

Figure 3: Program execution speed: 10^3 instructions per second.

4.1 Instruction mix

Instruction mix provides a coarse idea of what type of computations must be performed, and how much capability is to be assigned for a GPP. Figure 4 shows the instruction mix of each program. Instructions are classified into

five types: memory access (load and store), integer arithmetic and logical operation (ALO), floating-point operation, control transfer, and the remainder. This corresponds to the types of functional units used in the processor.

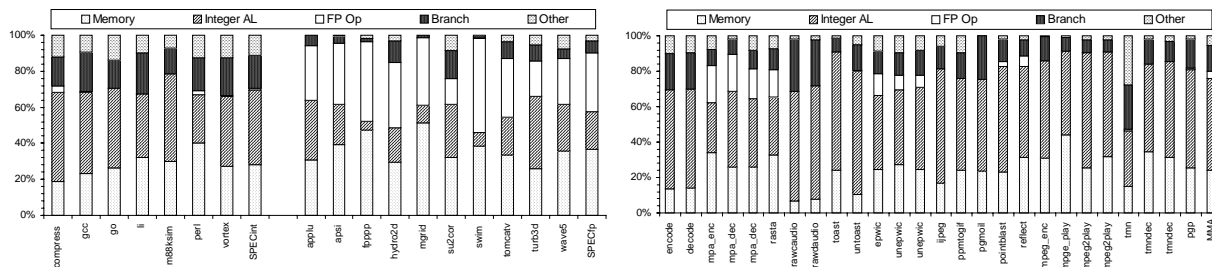


Figure 4: Instruction mix.

The memory access instruction ratio of more than two thirds of MMA programs is less than the average ratio of SPECint. Particularly, some audio programs such as *rawaudio*, *rawdaudio*, and *untoast* show very low memory access instruction ratios, while less variance is found among image programs for the same class of instructions. In SPECfp, memory access instructions are the most dominant type.

MMA programs are the most ALO-intensive: on average more than 56% of instructions executed are either integer or floating-point operations while they are only average 42% in SPECint. In the case of a 4-way superscalar operating at the peak rate, at least two integer ALOs need to be executed for MMA programs. Current multimedia instructions of GPPs, e.g., Intel’s MMX, are developed to take advantage of small integer operands of MMAs. However, we found considerable portions of floating-point operations in audio and image programs. Particularly, in MPEG audio programs, 56~77% of data items accessed from/to memory are 64-bit floating-point numbers (see Figure 12). In such cases, performance gain provided by the multimedia instructions cannot be significant.

Frequent control transfer (noted by *Branch* type in the graph) due to branch, jump, and procedure call and return instructions is one of the major obstacles to extracting higher ILP. SPECint shows the highest branch frequency whereas control transfer is especially rare in SPECfp. MMA programs are in the middle of the two. However, MPEG audio and video programs use very few branches. In MMA, more branches in highly repetitive loops can be eliminated at compile time. Low branch frequency results in larger basic block sizes, which is a positive factor in achieving higher ILP. A basic block is a sequence of instructions stored in contiguous locations without any control transfer in between.

The instruction mix of a program processing two input files varies by up to 6%. Interestingly, such a behavioral change occurs only in decoder programs, *mpa_dec*, *unepwic*, *mpeg2play*, and *tmndec*. For a given multimedia file, decoding is more likely to be used than encoding since once it is encoded, it is repeatedly decoded by end-users. In some pairs of encoder and decoder, a very similar instruction mix is observed for both.

MMA programs stress the ALO capability of the GPPs while less frequent memory access and branch are needed. Parallel processing on packed data types (data parallelism) using specialized instructions, e.g., the VIS, and/or hardware assist can compensate some of increased ALO requirement. However, the rest ALO still need be handled in conventional ways of exploiting ILP or thread level parallelism. The performance on integer operation is critical for MMAs as well. Therefore, for performance improvement on overall workload, additional integer units seem more beneficial than additional dedicated units for MMAs. Since unlike embedded processors, most GPPs include both integer and floating-point functional units, manual arithmetic conversion from floating-point into fixed-point sacrificing numerical accuracy cannot be always a good option.

4.2 Arithmetic operations

Instruction types that are most frequently used should have the shortest execution latency. Figure 5 depicts the distribution of arithmetic operations. The height of each column represents the sum of integer and floating-point operations in Figure 4. In MMA and SPECint, addition and subtraction are dominant among arithmetic operations. Multiplication is extremely infrequent in SPECint and its frequency in scientific programs, SPECfp, averages 13%.

In the literature, a very high frequency of multiply-accumulate (MAC) is often presumed for MMAs. The MAC-type multimedia instructions are also implemented on GPPs. However, we found that multiplication accounts for less than 10% of instructions executed in all programs except *toast*. Specifically, multiplication is not a major operation in video programs, unlike digital signal filtering algorithms where MAC is the most used operation. One reason for this is that optimizing MMA programs for a GPP results in reduction in MAC-type operations. Also, inner loops where MAC-type operations are performed (e.g., discrete cosine transform or DCT routines) are often overwhelmed

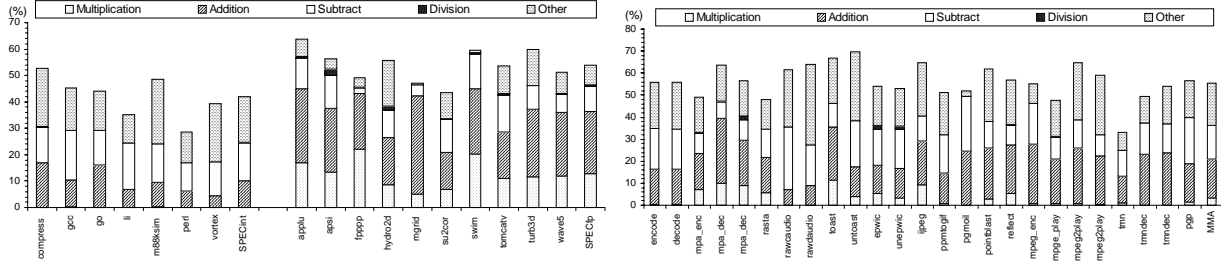


Figure 5: Distribution of integer and floating-point arithmetic operations.

by the rest of the code (e.g., file I/O, etc.). For example, for *mpeg_enc*, DCT routines account for about 13% of total runtime. Therefore, concentrating optimization efforts only on such operations cannot bring a big increase in multimedia performance on GPPs.

As expected, very few divisions are used in all application groups. In *mpeg_dec*, decoding a music file requires more divisions than decoding a speech file. Fortunately, MMA programs do not warrant changes in the current execution latency order of GPPs, the shortest for addition and subtraction and the longest for division. On an average, 19% of MMA instructions are logical, shift, other floating-point operations. A similar portion is observed for logical and shift operations in SPECint.

4.3 Branch

The branch behavior of a program significantly affects the pipeline performance of a GPP. Branch prediction is a commonly used technique in GPPs. The type and frequency of control transfer instructions is basic information for determining its proper features and estimating its performance. In Figure 6, we show the distribution of control transfers divided into *unconditional* and *conditional* branches. The lines represent the percentage of *taken* branches in the case of conditional branch instructions. The column labeled *unconditional* includes unconditional branches, jumps, calls, and returns.

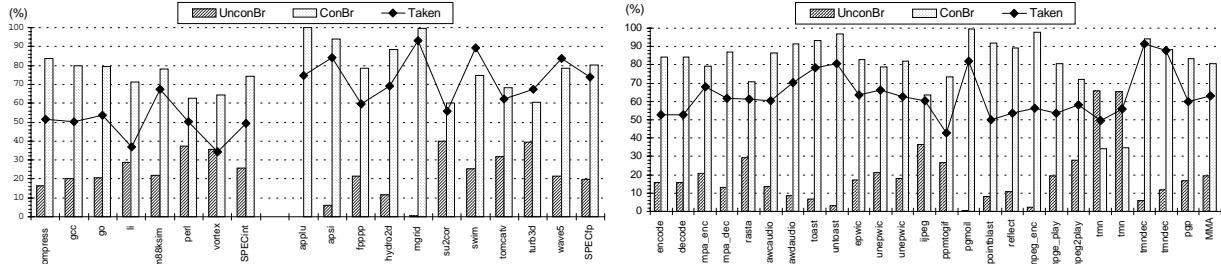
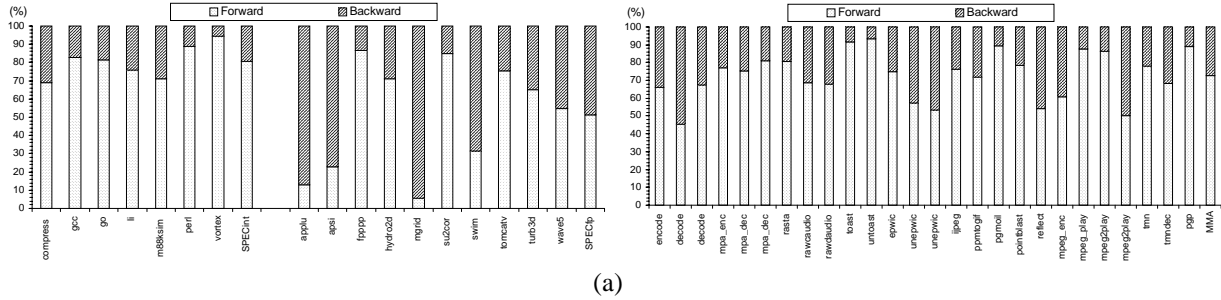


Figure 6: Distribution of branches.

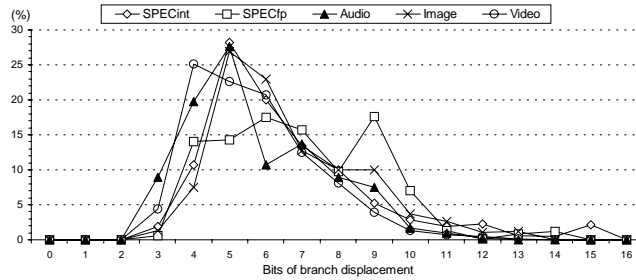
In MMA, a higher portion of branches on average is conditional compared to SPECint. However, MMA group has an exceptional case, *tmn*, which executes more unconditional than conditional branches. Branch penalty due to an unconditional branch can be overcome using hardware support. The graph shows that conditional branches in MMA are more likely to be taken. Infrequent branch instructions in SPECfp (Figure 4) are also most likely to cause control transfer. If branch is biased in either trend, even a simple branch prediction scheme can be very effective.

Some compilers may take the direction of branch instructions into account for branch prediction schemes. Figure 7a shows the direction of conditional branches. Forward conditional branches are dominant for SPECint and MMA programs. Backward branches are used in loops. We can assume that loop unrolling by the compiler removes some backward branches, which have a high probability of being taken. Interestingly, some SPECfp programs, *applu*, *apsi*, *mgrid*, and *swim*, change the program counter more in the backward direction. Even though conditional branches are more biased to “taken” in MMA, fluctuating behavior over different programs still makes the design of branch prediction schemes challenging.

The target address of a branch is obtained by adding an offset value to its program counter or other registers. This displacement is encoded in a field of the instruction. Consequently, how far a branch changes the program counter determines the number of bits required for the displacement field. Figure 7b shows the distribution of branch



(a)



(b)

Figure 7: (a) Direction of conditional branches. (a) Distribution of branch displacement.

displacement. The degree of displacement is represented by displacement bit $d (= \lceil \log_2 displacement \rceil)$ on the x-axis. As it is clear from the graph, branches are not issued to very near or far locations, and all the benchmark groups follow similar distribution lines. Therefore, MMA programs do not need special attention in this matter.

4.4 Basic block

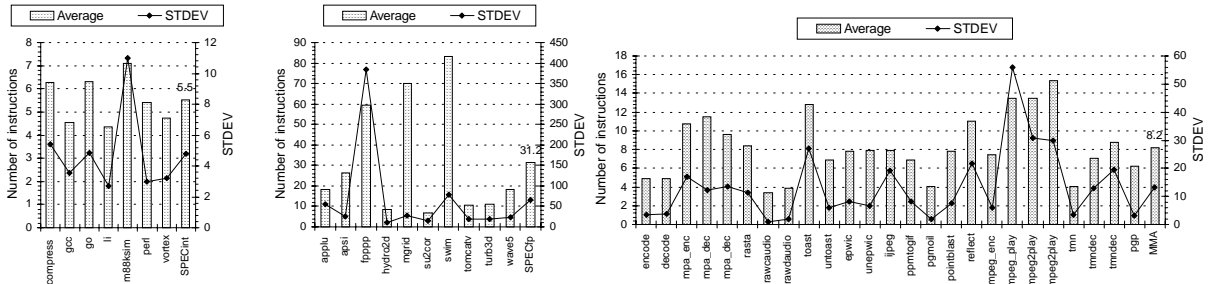


Figure 8: Average basic block size.

In order to achieve superscalar performance, higher ILP should be captured from instruction sequences. The compiler inspects each basic block to detect and eliminate data and structural dependences. The larger the basic block size, the higher the potential for ILP is expected. The basic block size can be used in determining the line size of the instruction cache or instruction fetch unit. Figure 8 and Figure 9 show basic block sizes and their distributions, respectively. Most integer programs depict a basic block size of 5 or 6. Lower branch frequencies of MMA programs, as shown in Figure 4, result in a small increase in the basic block size., e.g., by 3 instructions. However, some audio programs, *pgmoil*, and *tmn* still have small basic blocks. Different input files for some decoders also change average basic block size. On average, SPECfp shows a size of 31 instructions due to very few branches.

Note that standard deviations plotted on the right (secondary) y-axis are often higher than averages, meaning that the average values are not representative for the samples. This is because increased average basic block size is mostly determined by relatively small numbers of large basic blocks, say a basic block of more than 2^9 instructions. There is a one-to-one correspondence between Figure 8 and Figure 9: if the average basic block size of a program

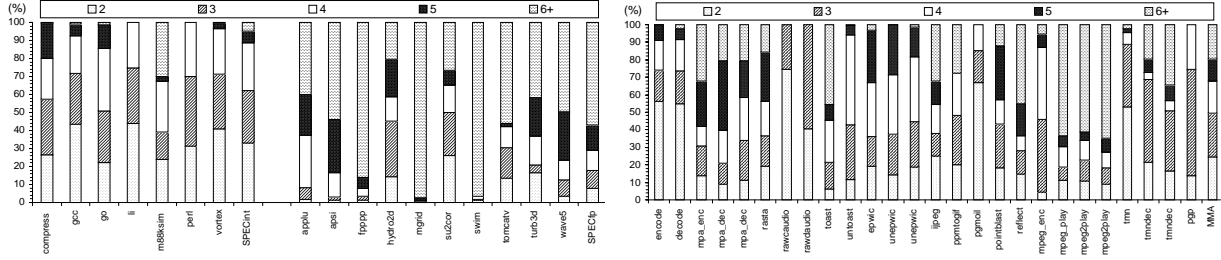


Figure 9: Distribution of basic blocks.

is large, a significant portion of instructions executed is from basic blocks of $b (> 2^6)$. Larger basic block sizes in MMA give insight into its higher degree of parallelism.

Since available ILP in a single basic block is very limited, multiple basic blocks that are likely to be executed consecutively are also examined simultaneously. As discussed earlier, if the processor can identify an unconditional branch and its target address in time, the next basic block can be executed with no penalty. We define this combination of basic blocks by unconditional branches as an *extended basic block*. On average, the size of an extended basic block is 7, 38, and 10 instructions for SPECint, SPECfp, and MMA, respectively. The distribution of extended basic blocks is also similar to Figure 9. Thus, we do not present it here.

4.5 Instruction reuse

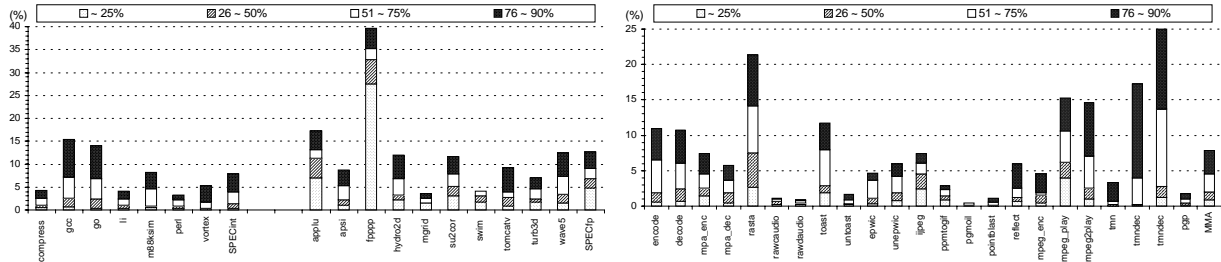


Figure 10: Portion of program that accounts for 25%, 50%, 75%, and 90% of instructions executed.

High locality in instruction accesses is common in GSAs, and thus instruction caches are very effective. Since MMAs execute the same set of operations on continuous data, we also expect high instruction reuse. Figure 10 shows what portion of a program accounts for what fraction of instructions executed. Although the distribution is more dynamic in MMA than in SPECint, both groups use only about 8% of code for 90% of instructions executed on average. Specially, all the image programs show very high instruction reuse. Two input files exhibit different results only in *tmndec*. The average of SPECfp is a bit higher, but few basic blocks are repeatedly executed.

Figure 11 shows what fractions of instructions executed are accounted for by the 10, 20, and 30 most frequently used basic blocks. On average, more than 60% of instructions executed comes from only 10 basic blocks in MMA and SPECfp. In comparison to SPECint, MMA shows higher basic block reuse. From this result, we can assume that the instruction cache of current GPPs can also provide a good performance on MMAs.

4.6 Data type and access pattern

Operands used in computations are initially loaded from memory. The size of data elements is used to determine optimal data path width and access cycle for communication between the processor and memory. Figure 12 shows the distribution of data access size by load and store instructions. The bottom four fields of each column represent integer data types: byte (8 bits), halfword (16 bits), word (32 bits), and doubleword (64 bits). Major data types used by the three groups are distinctive in most cases. For video programs, byte and halfword data are most frequently used. This is because the programs mostly manipulate 8-bit pixels. Audio programs use more halfword than byte data, but MPEG audio and *rasta* use larger integer data units, word and doubleword, as well as floating-point data.

References

- [1] M. Schlett, "Trends in embedded-microprocessor design," *Computer*, vol.31, no.8, pp. 44-9, August 1998.
- [2] J. Eyre and J. Bier, "DSP processors hit the mainstream," *Computer*, vol.31, no.8, pp. 51-59, August 1998.
- [3] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architectures for video compression-a survey," *Proceedings of the IEEE*, vol.83, no.2, pp. 220-246, February 1995.
- [4] S. Purcell, "The impact of Mpack 2," *IEEE Signal Processing Magazine*, vol.15, no.2, pp. 102-107, March 1998.
- [5] F. Sijstermans, E. J. Pol, B. Riemens, K. Vissers, S. Rathnam, and G. Slavenburg, "Design space exploration for future TriMedia CPUs," *Proc. IEEE Int'l Conf. Acoustics, Speech and Signal Processing*, pp. 3137-3140, May 1998.
- [6] R. B. Lee, Edited by: M. K. Ibrahim, P. Pirsch, and J. McCanny, "Multimedia extensions for general-purpose processors," *Proc. SiPS 97 Design and Implementation Formerly VLSI Signal Processing*, pp. 9-23, November 1997.
- [7] URL: <http://www.spec.org>.
- [8] R. H. Saavedra and A. J. Smith, "Analysis of benchmark characteristics and benchmark performance prediction," *ACM Transactions on Computer Systems*, vol.14, pp. 344-384, November 1996.
- [9] L. Kohn, G. Maturana, M. Tremblay, A. Prabhu, and G. Zyner, "The Visual Instruction Set (VIS) of the UltraSPARC microprocessor," *Elektronik Industrie*, vol.26, no.8, pp. 68-71, August 1995.
- [10] J. Hanks, "Improved performance with MMX technology," *Image Processing*, pp. 8-10, November 1997.
- [11] V. Lappalainen, "Performance analysis of Intel MMX technology for an H.263 video encoder," *Proc. 6th ACM Int'l Conf. Multimedia*, pp. 309-314, September 1998.
- [12] R. Bhargava, L. K. John, B. L. Evans, and R. Radhakrishnan, "Evaluating MMX Technology Using DSP and Multimedia Applications," *Proc. 31st ACM/IEEE Int'l Symp. Microarchitecture*, pp. 37-46, December 1998.
- [13] P. Ranganathan, S. Adve, and N. Jouppi, "Performance of image processing with general-purpose processors and media ISA extensions," *Proc. 26th ACM Int'l Symp. Computer Architecture*, pp. 124-135, May 1999.
- [14] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, vol.16, pp. 42-50, August 1996.
- [15] H. Balakrishnan and R. Garg, "Multimedia SPECmarks: a performance comparison of multimedia programs on different architectures," *UCB EECS Technical Report CSD-96-926*, 1996.
- [16] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," *Proc. IEEE/ACM Int'l Conf. Microarchitecture*, pp. 330-335, December 1997.
- [17] D. L. Gall, "MPEG: a video compression standard for multimedia applications," *Communications of the ACM*, vol.34, no.4, pp. 46-58, April 1991.
- [18] URL: <http://www.spec.org/gpc/mbc.static/>.
- [19] A. Zandi and S. I. Sudharsanan, "Benchmarking multimedia performance," *Proceedings of the SPIE - The International Society for Optical Engineering*, vol.3311, pp. 8-13, January 1998.
- [20] L. Heng and A. Wolfe, "Available parallelism in video applications," *Proc. IEEE/ACM Int'l Conf. Microarchitecture*, pp. 321-329, December 1997.
- [21] B. Cmelik and D. Keppel, "Shade: a fast instruction-set simulator for execution profiling," *Performance Evaluation Review*, vol. 22, pp. 128-137, May 1994.